Regular Expressions

Reading

Section 1.3

Practice problems (page 85): 1.18, 1.19, 1.28

Optional: 1.21

Regular Expressions

Regular expressions are an expressive way to describe languages, especially languages that contain repetitions of patterns. Regular expressions are built out of primitive components.

A **regular expression** R has one of the following forms:

- The symbol a where a is a letter from the alphabet Σ, representing a language that matches only the string on that one symbol.
- The symbol ϵ, representing a language that matches just the empty string.
- $\emptyset$, representing a language that matches nothing.
- $R_1 \cup R_2$, or also $R_1|R_2$, where R_1, R_2 are already-formed regular expressions. Represents a language containing the strings of the language of R_1 as well as those of the language of R_2.
- $R_1 \circ R_2$, or also R_1R_2, where R_1, R_2 are already-formed regular expressions. Represents a language containing the concatenations of strings of the languages represented by R_1 and R_2.
- R^*, where R is a regular expression. Represents the Kleene star of the language represented by R.

This is perhaps the first example of a **recursive or inductive definition**, where a new item of a certain type can be built from previously-constructed items. We of course have already seen this idea in OCAML's type system.

For example, we can denote the language that contains any string on the letters a, b, c, but that starts with either a or b, by:

$$(a|b)(a|b|c)^*$$

Regular expressions are available on almost any programming language, and they are used when scanning through text. The rules for what a token/valid-language-element constitutes on most programming languages can be expressed via regular expressions. Regular expressions used in practice usually involve a number of conveniences, including:

- Notation for ranges, e.g. $[a-z]$ denotes all letters
- Special escape sequences to denote for example "all whitespace characters"
- A "plus" operator to indicate at least one occurence of something (the Kleene star allows 0 occurences)
- A "question mark" operator to indicate the optional matching of an item.
- Storing parts of a "match" in a variable that can then be referenced later on.
- Special character for indicating matching "a single input of anything". You can think of this as a union of all the elements.

The remarkable fact that we will build towards is that regular expressions have the same expressive power as DFAs/NFAs:

> A language is regular, i.e. recognized by a DFA/NFA, if and only if it can be expressed via a regular expression.

There are two directions to this, and one is easy:

> Given a regular expression describing a language, we can construct a corresponding NFA recognizing that same language. So any language represented by a regular expression is a regular language.

This is easy to see:

- We have already seen in our section on NFAs how to create NFAs/DFAs describing the empty language, the language containing just the empty string, and the language matching exactly the string consisting of one specific element. These are our main building blocks.
- We have also seen how given two NFAs for two languages, we can form NFAs for their union and their concatenation, as well as how to form the NFA for the Kleene star of a language. This covers all the cases for regular expression.

Exercise: Use this idea to build the NFA for the regular expression (a\cup b)^∗aba(a\cup b)^∗. Make sure to follow the general instructions for concatenation, without taking shortcuts.

The converse is also true but harder to show:

> For every regular language there is a regular expression describing it.

We will skip the proof of this, but those interested can find the details on pages 69 through 76. The key step in the proof is the introduction of the concept of **generalized nondeterministic finite automata**, where transitions are "tagged" by a regular expression, and you follow that transition by consuming a part of the input taht matches that expression. An initial such automaton is created trivially from an NFA that recognizes that language, and that NFA is then slowly contracted one state at a time, at the same time making the transition regular expressions more complex. The final result consists effectively of a start state and an accept state, connected via a regular expression that then describes the language.